

CMPT 280

Topic 8: Tree Traversals

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 8

Depth First Traversals

Recall the general algorithm for depth-first traversals:

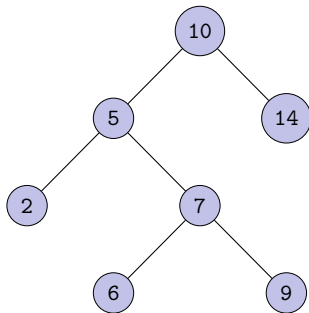
```
1 Algorithm depthFirstTraversal(N)
2 Parameters:
3     N is a tree node
4
5 visit node N
6 for each child T of N // Assume left to right, though it need not be so.
7     depthFirstTraversal(T)
```

Exercise 1

- Write a method for `LinkedSimpleTree280<I>` that performs a depth-first traversal that prints out the contents of each node.
- Besides being a depth-first traversal, what kind of binary tree traversal is your solution?

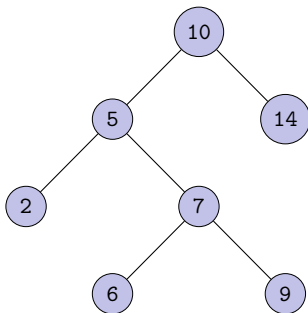
In-order Traversal of Binary Search Trees

- Recall the Binary Search Trees (from CMPT 115).
- What special property does the in-order traversal of a binary search tree have?



Exercise 2: In-order traversals

- Modify the method from Exercise 1 to create an in-order traversal.



Exercise 3: Post-order traversals

- Modify the method from Exercise 1 to create a post-order traversal.
- The sequence generated by a post-order traversal has no special significance for binary search trees.
- BUT... we can do some fairly interesting things with post-order traversal that don't involve printing the node contents.

Post-order traversals

We can accomplish the following with depth-first traversals:

- Count number of nodes in the tree.
- Determine height of the tree
- Evaluate expression tree.

Exercise 4: Counting number of nodes.

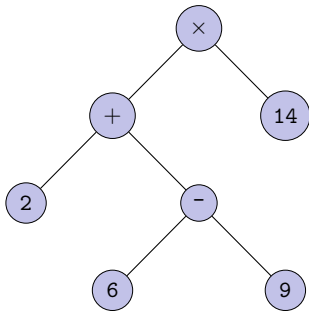
- Write a post-order traversal for `LinkedSimpleTree280<I>` that returns the number of nodes in the tree.
- Hint: If you know how many nodes are in the left and right subtrees, then the number of nodes in the tree is $1 + \# \text{ nodes in left subtree} + \# \text{ nodes in right subtree}$.
- What was the "visit" operation in the solution?

Exercise 5: Computing tree height.

- Modify the method from Exercise 4 to compute the height of the tree. It shouldn't take much since both are post-order traversals!
- Hint: If you know the height of the left and right subtrees, then the height of the tree is $1 + \max(\text{left subtree height}, \text{right subtree height})$;
- What was the "visit" operation in the solution?

Postorder Traversal: Evaluate Expression Tree

- Why is evaluating an expression tree a post-order traversal?



Breadth-First Traversals

- Algorithm for BFT requires a queue.
- Idea:
 - After we visit a node, put all of its children on a queue.
 - Keep visiting nodes as long as the queue is not empty.

```
1 Algorithm breathFirstTraversal(T)
2 T is a tree.
3
4 Let Q be a queue.
5 Q.add(T.rootNode)
6
7 while(Q is not empty)
8     p = Q.remove();
9     visit p
10    for each child s of p
11        Q.add(s)
```

Exercise 6

- Write a method for `LinkedSimpleTree280` that prints the contents of the nodes in level-order. The method should follow the BFT algorithm below with an appropriate "visit" operation.
- Recall: printing the nodes of a tree in level-order is a special case of a breath-first traversal. Level-order traversal visits the nodes on each level from left to right, starting at level 0.
- Hint: for a queue we could use any of `java.util.LinkedList`, `LinkedList280<I>`, or `LinkedQueue280<I>`.

```
1 Algorithm breathFirstTraversal(T)
2   T is a tree.
3
4   Let Q be a queue.
5   Q.add(T.rootNode)
6
7   while(Q is not empty)
8     p = Q.remove();
9     visit p
10    for each child s of p
11      Q.add(s)
```

Deep Clone: A depth-first algorithm.

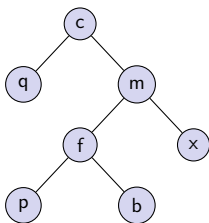
Deep cloning of a object can be performed by a depth-first traversal of the reference fields in an object. Note that the references in the object X and the objects those objects reference (etc.) form a tree (whether X is actually a tree ADT or not)!

```
1 Algorithm deepClone(X)
2 X is an object
3
4 Y = shallowClone(X) // "Visit" operation is a shallow clone of
5                      // the current object.
6
7 for each non-null reference X.r in X
8     Y.r = deepClone(X.r);
9
10 return Y
```

Can we re-write this algorithm so that it is a bit more specific to cloning a `LinkedSimpleTree280<I>`?

Printing the Structure of Binary Trees

- Trees are hard to draw with root at the top.
- Trees are easy to draw with root at the left and descendants to the right:



```
1: c
  2: q
    3: f
      4: p
    3: f
      4: b
  2: m
    3: x
```

- Right subtree of a node is above and indented; left subtree is below and indented.
- This can be achieved with a non-standard traversal:
right, root, left

Printing the Structure of Trees

(This is in `LinkedSimpleTree280`)

```
1  /** Form a string representation that includes level numbers when the tree
2   *   is at level i.
3   *   @param i   the level of this tree in the main tree, i.e., number of
4   *               indentations minus 1
5   *   @timing    Time = O(n), where n = number of items in this tree */
6   protected String toStringByLevel(int i)
7   {
8       StringBuffer blanks = new StringBuffer((i - 1) * 5);
9       for (int j = 1; j < i; j++)
10           blanks.append("      ");
11
12       if (isEmpty())
13           return  "\n" + blanks + i + ":_";
14
15       else
16       {
17           boolean printSubtrees = !rootLeftSubtree().isEmpty() ||
18                                   !rootRightSubtree().isEmpty();
19
20           String result = "";
21           if (printSubtrees)
22               result += rootRightSubtree().toStringByLevel(i+1);
23
24           result += "\n" + blanks + i + ":_" + rootItem();
25
26           if (printSubtrees)
27               result += rootLeftSubtree().toStringByLevel(i+1);
28
29           return result;
30       }
31 }
```


Next Class

- **Next class reading:** Chapter 9: Ordered Binary Trees